







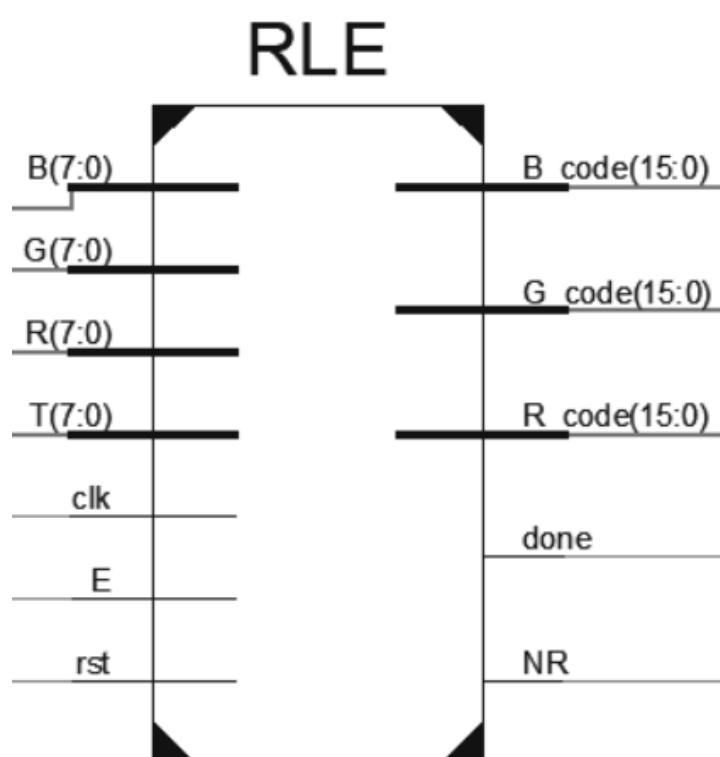
به نام خدا

پیاده‌سازی الگوریتم فشرده‌سازی RLE بر بستر FPGA

فایل‌های ورودی و خروجی برنامه به همراه تصاویر اولیه و نتایج در پوشه test قرار دارند. ابعاد تصاویر ورودی باید ۱۲۸ در ۱۲۸ پیکسل باشد.

تصویر ورودی	تصویر خروجی $T = 30$
	
	
	

نمای شماتیک RLE



RLE كد

```

module RLE(rst, clk, NR, E, T, R, G, B, R_code, G_code, B_code, done);
    input rst, clk, E;
    input [7:0] T;
    input [7:0] R, G, B;
    output reg [15:0] R_code, G_code, B_code;
    output reg done, NR;

    reg[7:0] r_cnt, g_cnt, b_cnt;
    reg[8:0] r_cur, g_cur, b_cur;
    reg [8:0] i, j;
    parameter [8:0] MAX_CUR = 511;
    parameter [6:0] MAX_I = 127;

    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            R_code = 0;
            G_code = 0;
            B_code = 0;
            done = 0;
            r_cnt = 0;
            g_cnt = 0;
            b_cnt = 0;
            r_cur = MAX_CUR;
            g_cur = MAX_CUR;
            b_cur = MAX_CUR;
            i = 0;
            j = 0;
            NR = 0;
        end
        else begin
            NR = 0;
            if (E) begin
                if (r_cur == MAX_CUR) begin
                    R_code = 0;
                    r_cnt = 1;
                    r_cur = R;
                end
                else if (i == MAX_I && ((R-r_cur >= 0 && R-r_cur <= T) || (r_cur-R >= 0 && r_cur-R <= T))) begin
                    r_cnt = r_cnt + 1;
                    R_code = {r_cnt, r_cur[7:0]};
                    r_cnt = 0;
                    r_cur = MAX_CUR;
                end
                else if ((R-r_cur >= 0 && R-r_cur <= T) || (r_cur-R >= 0 && r_cur-R <= T)) begin
                    r_cnt = r_cnt + 1;
                    R_code = 0;
                end
                else begin
                    R_code = {r_cnt, r_cur[7:0]};
                    r_cnt = 1;
                    r_cur = R;
                end
                end
            if (g_cur == MAX_CUR) begin
                G_code = 0;
            end
        end
    end

```

```

    g_cnt = 1;
    g_cur = G;
end
else if (i == MAX_I && ((G-g_cur >= 0 && G-g_cur <= T) || (g_cur-G >= 0 && g_cur-G <= T))) begin
    g_cnt = g_cnt + 1;
    G_code = {g_cnt, g_cur[7:0]};
    g_cnt = 0;
    g_cur = MAX_CUR;
end
else if ((G-g_cur >= 0 && G-g_cur <= T) || (g_cur-G >= 0 && g_cur-G <= T)) begin
    g_cnt = g_cnt + 1;
    G_code = 0;
end
else begin
    G_code = {g_cnt, g_cur[7:0]};
    g_cnt = 1;
    g_cur = G;
end

if (b_cur == MAX_CUR) begin
    B_code = 0;
    b_cnt = 1;
    b_cur = B;
end
else if (i == MAX_I && ((B-b_cur >= 0 && B-b_cur <= T) || (b_cur-B >= 0 && b_cur-B <= T))) begin
    b_cnt = b_cnt + 1;
    B_code = {b_cnt, b_cur[7:0]};
    b_cnt = 0;
    b_cur = MAX_CUR;
end
else if ((B-b_cur >= 0 && B-b_cur <= T) || (b_cur-B >= 0 && b_cur-B <= T)) begin
    b_cnt = b_cnt + 1;
    B_code = 0;
end
else begin
    B_code = {b_cnt, b_cur[7:0]};
    b_cnt = 1;
    b_cur = B;
end

i = i + 1;
if (i > MAX_I)
    j = j + 1;
end
else begin
    if (j > MAX_I) begin
        j = j + 1;
        if (j >= MAX_I + 8)
            done = 1;
        end
    if (i == MAX_I + 1) begin
        if (r_cur == MAX_CUR)
            R_code = 0;
        else
            R_code = {r_cnt, r_cur[7:0]};
            r_cnt = 0;
            r_cur = MAX_CUR;

            if (g_cur == MAX_CUR)

```

```

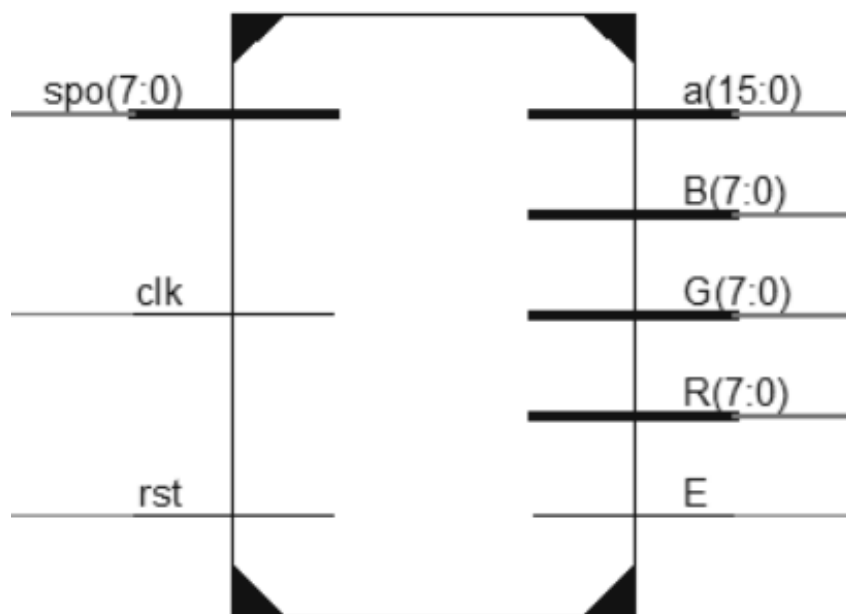
        G_code = 0;
    else
        G_code = {g_cnt, g_cur[7:0]};
        g_cnt = 0;
        g_cur = MAX_CUR;

        if (b_cur == MAX_CUR)
            B_code = 0;
        else
            B_code = {b_cnt, b_cur[7:0]};
            b_cnt = 0;
            b_cur = MAX_CUR;

        NR = 1;
        i = 0;
    end
else begin
    R_code = 0;
    G_code = 0;
    B_code = 0;
end
end
end
end
endmodule

```

RLE_RAM_Interface



RLE RAM Interface كد

```
module RLE_RAM_Interface(clk, rst, a, spo, E, R, G, B);
    input clk, rst;
    input [7:0] spo;
    output reg E;
    output reg [15:0] a;
    output reg [7:0] R, G, B;

    parameter HOLD_CLK = 1;
    reg [15:0] hold_cnt;

    parameter RBA = 0, GBA = 16384, BBA = 32768;
    reg [14:0] offset;

    reg [2:0] state;
    parameter [2:0] r1 = 0, r2 = 1, g1 = 2, g2 = 3, b1 = 4, b2 = 5, hold = 6;

    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            E = 0;
            R = 0;
            G = 0;
            B = 0;
            offset = 0;
            state = r1;
            hold_cnt = 0;
        end
        else begin
            E = 0;
            if (offset < GBA) begin
                if (state == r1) begin
                    a = offset + RBA;
                    state = r2;
                end
                else if (state == r2) begin
                    R = spo;
                    state = g1;
                end

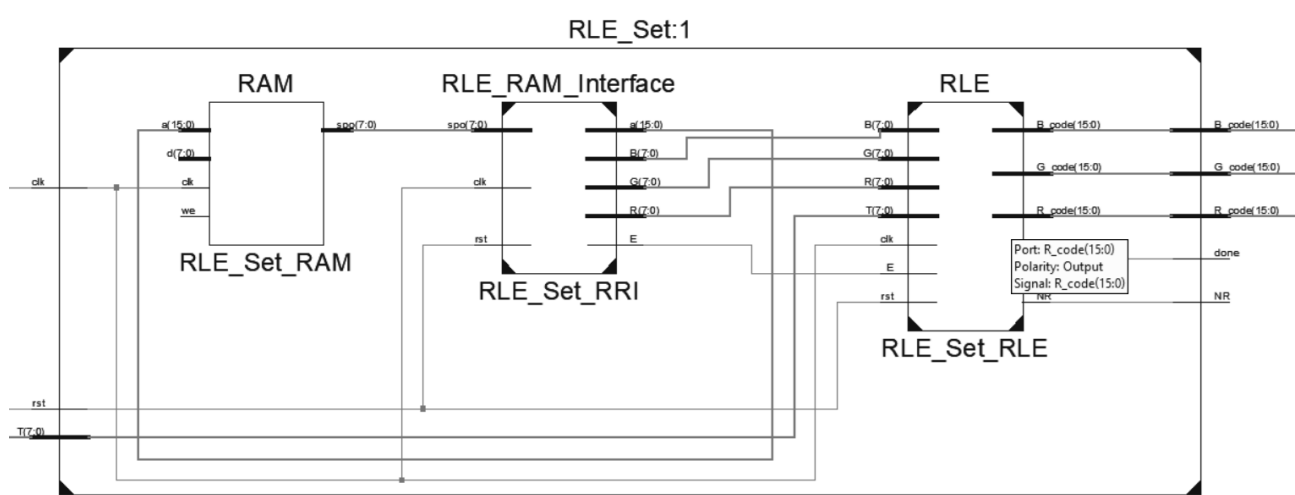
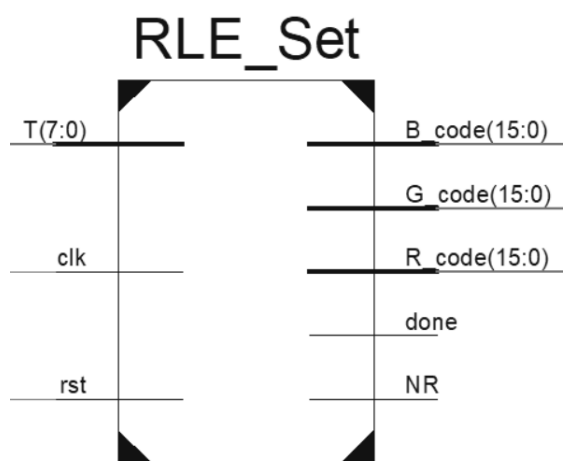
                else if (state == g1) begin
                    a = offset + GBA;
                    state = g2;
                end
                else if (state == g2) begin
                    G = spo;
                    state = b1;
                end

                else if (state == b1) begin
                    a = offset + BBA;
                    state = b2;
                end
                else if (state == b2) begin
                    B = spo;
                    E = 1;
                    offset = offset + 1;
                    state = hold;
                end
            end
        end
    end
endmodule
```

```
    end

    else if (state == hold) begin
        hold_cnt = hold_cnt + 1;
        if (hold_cnt >= HOLD_CLK) begin
            hold_cnt = 0;
            state = r1;
        end
    end
end
end
end
end
endmodule
```


نمای شماتیک RLE Set



RLE Set كد

```
module RLE_Set(clk, rst, T, done, NR, R_code, G_code, B_code);
    input clk, rst;
    input [7:0] T;
    output done, NR;
    output [15:0] R_code, G_code, B_code;

    parameter HOLD_CLK = 1;

    wire [15:0] a;
    wire [7:0] spo;
    wire E;
    wire [7:0] R, G, B;

    RAM RLE_Set_RAM (
        .a(a),
        .clk(clk),
        .spo(spo)
    );

    RLE_RAM_Interface RLE_Set_RRI (
        .rst(rst),
        .clk(clk),
        .a(a),
        .spo(spo),
        .E(E),
        .R(R),
        .G(G),
        .B(B)
    );
    defparam RLE_Set_RRI.HOLD_CLK = HOLD_CLK;

    RLE RLE_Set_RLE (
        .rst(rst),
        .clk(clk),
        .E(E),
        .T(T),
        .R(R),
        .G(G),
        .B(B),
        .R_code(R_code),
        .G_code(G_code),
        .B_code(B_code),
        .done(done),
        .NR(NR)
    );
endmodule
```

RLE Set تست

```
module RLE_Set_tb;
  reg clk, rst;
  reg [7:0] T;

  wire [15:0] R_code, G_code, B_code;
  wire done, NR;
  integer rf, gf, bf;

  RLE_Set uut (
    .clk(clk),
    .rst(rst),
    .T(T),
    .done(done),
    .NR(NR),
    .R_code(R_code),
    .G_code(G_code),
    .B_code(B_code)
  );

  initial begin
    clk = 0;
    rst = 0;
    T = 0;
    #2 rst = 1; #7 rst = 0;
  end

  initial forever #10 clk = ~clk;

  initial begin
    #20;

    rf = $fopen("test/encoded/R.txt", "w");
    gf = $fopen("test/encoded/G.txt", "w");
    bf = $fopen("test/encoded/B.txt", "w");

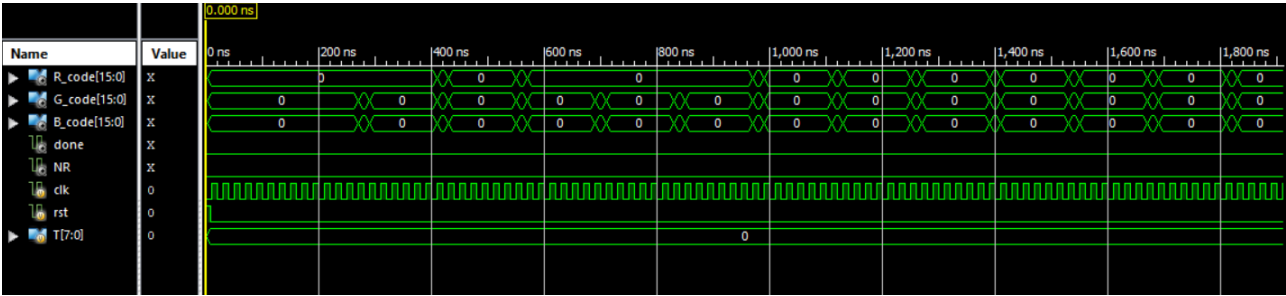
    while (~done) begin
      #20;

      if (R_code > 0)
        $fwrite(rf, "%0d,", R_code);
        //$fwrite(rf, "%0d,%0d ", R_code[15:8], R_code[7:0]);
      if (NR)
        $fwrite(rf, "\n");

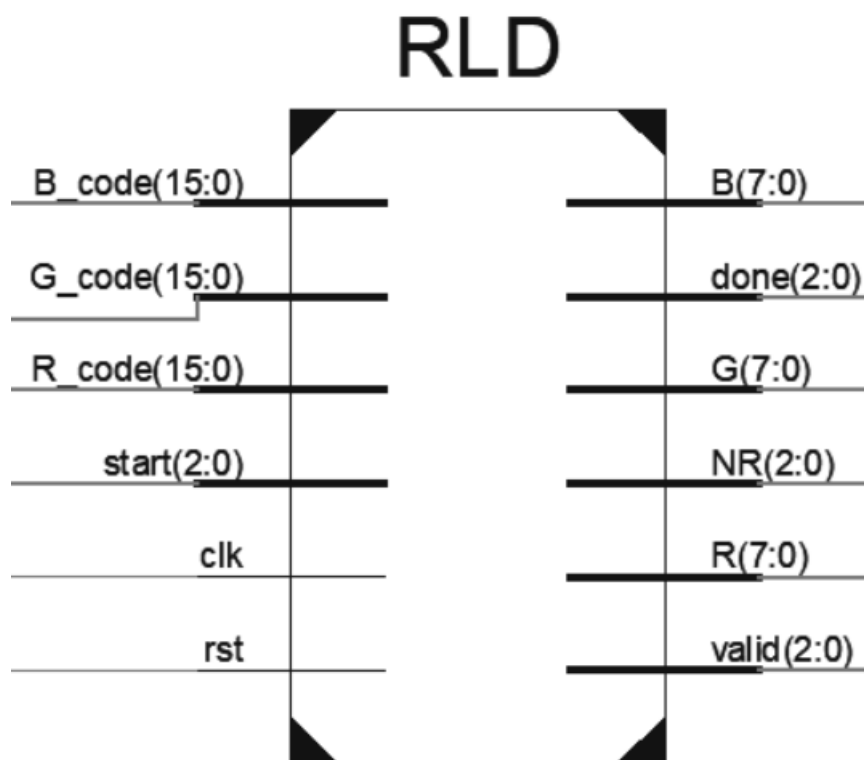
      if (G_code > 0)
        $fwrite(gf, "%0d,", G_code);
        //$fwrite(gf, "%0d,%0d ", G_code[15:8], G_code[7:0]);
      if (NR)
        $fwrite(gf, "\n");

      if (B_code > 0)
        $fwrite(bf, "%0d,", B_code);
        //$fwrite(bf, "%0d,%0d ", B_code[15:8], B_code[7:0]);
      if (NR)
        $fwrite(bf, "\n");
    end
  end
```

```
$fclose(rf);
$fclose(gf);
$fclose(bf);
rf = 0;
gf = 0;
bf = 0;
end
endmodule
```



نمای شماتیک RLD



```

module RLD(rst, clk, start, R_code, G_code, B_code, R, G, B, done, valid, NR);
    input rst, clk;
    input [2:0] start;
    input [15:0] R_code, G_code, B_code;
    output reg [7:0] R, G, B;
    output reg [2:0] valid, NR, done;

    parameter [6:0] MAX_I = 127;

    reg [15:0] r_c, r_din;
    reg [7:0] r_cnt, r_cur;
    reg r_wr, r_rd;
    reg [7:0] r_i, r_j;
    reg [1:0] r_k;
    wire r_full, r_empty, r_valid;
    wire [15:0] r_dout;

    reg [15:0] g_c, g_din;
    reg [7:0] g_cnt, g_cur;
    reg g_wr, g_rd;
    reg [7:0] g_i, g_j;
    reg [1:0] g_k;
    wire g_full, g_empty, g_valid;
    wire [15:0] g_dout;

    reg [15:0] b_c, b_din;
    reg [7:0] b_cnt, b_cur;
    reg b_wr, b_rd;
    reg [7:0] b_i, b_j;
    reg [1:0] b_k;
    wire b_full, b_empty, b_valid;
    wire [15:0] b_dout;

    BuffR_BUF (
        .clk(clk),
        .rst(rst),
        .din(r_din),
        .wr_en(r_wr),
        .rd_en(r_rd),
        .dout(r_dout),
        .full(r_full),
        .empty(r_empty),
        .valid(r_valid)
    );

    BuffG_BUF (
        .clk(clk),
        .rst(rst),
        .din(g_din),
        .wr_en(g_wr),
        .rd_en(g_rd),
        .dout(g_dout),
        .full(g_full),
        .empty(g_empty),
        .valid(g_valid)
    );

```

```

);

Buff B_BUF (
    .clk(clk),
    .rst(rst),
    .din(b_din),
    .wr_en(b_wr),
    .rd_en(b_rd),
    .dout(b_dout),
    .full(b_full),
    .empty(b_empty),
    .valid(b_valid)
);

always @ (posedge clk or posedge rst) begin
    if (rst) begin
        r_din <= 0;
        r_wr <= 0;

        g_din <= 0;
        g_wr <= 0;

        b_din <= 0;
        b_wr <= 0;
    end
    else begin
        if (R_code > 0 && ~r_full && start[0]) begin
            r_din <= R_code;
            r_wr <= 1;
        end
        else begin
            r_wr <= 0;
        end

        if (G_code > 0 && ~g_full && start[1]) begin
            g_din <= G_code;
            g_wr <= 1;
        end
        else begin
            g_wr <= 0;
        end

        if (B_code > 0 && ~b_full && start[2]) begin
            b_din <= B_code;
            b_wr <= 1;
        end
        else begin
            b_wr <= 0;
        end
    end
end

always @ (posedge clk or posedge rst) begin
    if (rst) begin
        done = 0;
        valid = 0;
        NR = 0;
        R = 0;
        G = 0;
    end
end

```

```

B = 0;

r_rd = 0;
r_c = 0;
r_cur = 0;
r_cnt = 0;
r_i = 0;
r_j = 0;
r_k = 0;

g_rd = 0;
g_c = 0;
g_cur = 0;
g_cnt = 0;
g_i = 0;
g_j = 0;
g_k = 0;

b_rd = 0;
b_c = 0;
b_cur = 0;
b_cnt = 0;
b_i = 0;
b_j = 0;
b_k = 0;
end
else begin

// R Section
if (r_cnt == 0 && r_c == r_dout && ~r_empty && ~r_wr) begin
    r_rd = 1;
    if (r_i == 0)
        r_c = 0;
    end
    else begin
        r_rd = 0;
    end

    if (r_i == 0)
        if (r_k == 3)
            r_k = 0;
        else
            r_k = r_k + 1;
        end

    if (r_cnt == 0 && r_dout > 0 && r_c != r_dout && r_valid && r_k == 0) begin
        r_c = r_dout;
        r_cnt = r_dout[15:8];
        r_cur = r_dout[7:0];
    end

    if (r_cnt > 0) begin
        R = r_cur;
        valid[0] = 1;
        r_cnt = r_cnt - 1;
        r_i = r_i + 1;
    end
    else begin
        valid[0] = 0;
    end
end

```



```

if (r_i == MAX_I + 1 && r_cnt == 0) begin
    r_j = r_j + 1;
    NR[0] = 1;
    r_i = 0;
end
else begin
    NR[0] = 0;
end

if (r_j == MAX_I + 1)
    done[0] = 1;

// G Section
if (g_cnt == 0 && g_c == g_dout && ~g_empty && ~g_wr) begin
    g_rd = 1;
    if (g_i == 0)
        g_c = 0;
end
else begin
    g_rd = 0;
end

if (g_i == 0)
    if (g_k == 3)
        g_k = 0;
    else
        g_k = g_k + 1;

if (g_cnt == 0 && g_dout > 0 && g_c != g_dout && g_valid && g_k == 0) begin
    g_c = g_dout;
    g_cnt = g_dout[15:8];
    g_cur = g_dout[7:0];
end

if (g_cnt > 0) begin
    G = g_cur;
    valid[1] = 1;
    g_cnt = g_cnt - 1;
    g_i = g_i + 1;
end
else begin
    valid[1] = 0;
end

if (g_i == MAX_I + 1 && g_cnt == 0) begin
    g_j = g_j + 1;
    NR[1] = 1;
    g_i = 0;
end
else begin
    NR[1] = 0;
end

if (g_j == MAX_I + 1)
    done[1] = 1;

// B Section
if (b_cnt == 0 && b_c == b_dout && ~b_empty && ~b_wr) begin

```

```

        b_rd = 1;
        if (b_i == 0)
            b_c = 0;
        end
    else begin
        b_rd = 0;
    end

    if (b_i == 0)
        if (b_k == 3)
            b_k = 0;
        else
            b_k = b_k + 1;
        end

        if (b_cnt == 0 && b_dout > 0 && b_c != b_dout && b_valid && b_k == 0) begin
            b_c = b_dout;
            b_cnt = b_dout[15:8];
            b_cur = b_dout[7:0];
        end

        if (b_cnt > 0) begin
            B = b_cur;
            valid[2] = 1;
            b_cnt = b_cnt - 1;
            b_i = b_i + 1;
        end
    else begin
        valid[2] = 0;
    end

    if (b_i == MAX_I + 1 && b_cnt == 0) begin
        b_j = b_j + 1;
        NR[2] = 1;
        b_i = 0;
    end
    else begin
        NR[2] = 0;
    end

    if (b_j == MAX_I + 1)
        done[2] = 1;

    end
end
endmodule

```

RLD تست

```
module RLD_tb;
  reg rst, clk;
  reg [2:0] start;
  reg [15:0] R_code, G_code, B_code;

  wire [7:0] R, G, B;
  wire [2:0] valid, NR, done;

  integer rrf, rgf, rbf;
  integer wrf, wgf, wbf;

  RLD uut (
    .rst(rst),
    .clk(clk),
    .start(start),
    .R_code(R_code),
    .G_code(G_code),
    .B_code(B_code),
    .R(R),
    .G(G),
    .B(B),
    .done(done),
    .valid(valid),
    .NR(NR)
  );

  initial begin
    rst = 0;
    clk = 0;
    start = 0;
    R_code = 0;
    G_code = 0;
    B_code = 0;
    #2 rst = 1; #7 rst = 0;
  end

  initial forever #10 clk = ~clk;

  initial begin
    #100;

    rrf = $fopen("test/encoded/R.txt", "r");
    $fscanf(rrf, "%d", R_code);
    start[0] = 1;
    #20;
    start[0] = 0;

    while (~R_code && ~done[0]) begin
      #600;
      $fscanf(rrf, "%d", R_code);
      start[0] = 1;
      #20;
      start[0] = 0;
    end
  end
```

```

    $fclose(rff);
    rrf = 0;
end

initial begin
    #100;

    rgf = $fopen("test/encoded/G.txt", "r");
    $fscanf(rgf, "%d", G_code);
    start[1] = 1;
    #20;
    start[1] = 0;

    while (~G_code && ~done[1]) begin
        #600;
        $fscanf(rgf, "%d", G_code);
        start[1] = 1;
        #20;
        start[1] = 0;
    end

    $fclose(rgf);
    rgf = 0;
end

initial begin
    #100;

    rbf = $fopen("test/encoded/B.txt", "r");
    $fscanf(rbf, "%d", B_code);
    start[2] = 1;
    #20;
    start[2] = 0;

    while (~B_code && ~done[2]) begin
        #600;
        $fscanf(rbf, "%d", B_code);
        start[2] = 1;
        #20;
        start[2] = 0;
    end

    $fclose(rbf);
    rbf = 0;
end

initial begin
    #100;

    wrf = $fopen("test/decoded/R.txt", "w");

    while (~done[0]) begin
        #20;
        if (valid[0])
            $fwrite(wrf, "%0d", R);
        if (NR[0])
            $fwrite(wrf, "\n");
    end
end

```

```

    $fclose(wrf);
    wrf = 0;
end

initial begin
    #100;

    wgf = $fopen("test/decoded/G.txt", "w");

    while (~done[1]) begin
        #20;
        if (valid[1])
            $fwrite(wgf, "%0d,", G);
        if (NR[1])
            $fwrite(wgf, "\n");
        end

    $fclose(wgf);
    wgf = 0;
end

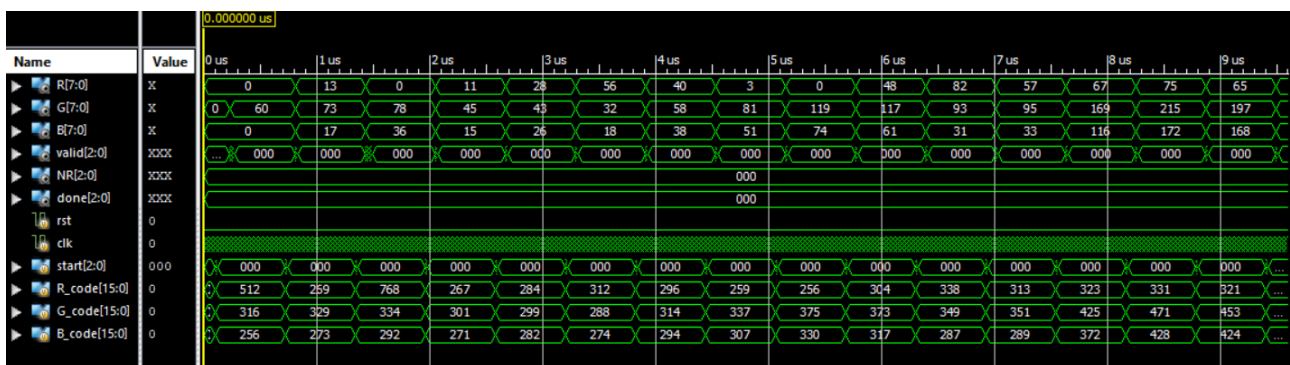
initial begin
    #100;

    wbf = $fopen("test/decoded/B.txt", "w");

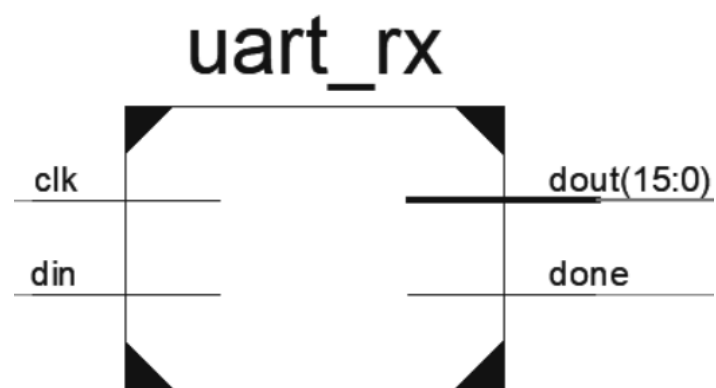
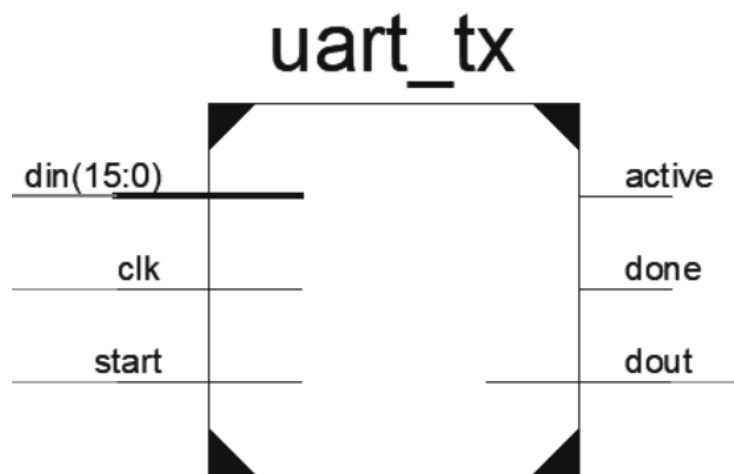
    while (~done[2]) begin
        #20;
        if (valid[2])
            $fwrite(wbf, "%0d,", B);
        if (NR[2])
            $fwrite(wbf, "\n");
        end

    $fclose(wbf);
    wbf = 0;
end
endmodule

```



نمای شماتیک Uart



Uart Transmitter ك

```

module uart_tx
(
    input    clk,
    input    start,
    input [15:0] din,
    output    active,
    output reg dout,
    output    done
);

// 50000000 / 115200 = 435
parameter CLKS_PER_BIT = 435;

parameter s_IDLE      = 3'b000;
parameter s_TX_START_BIT = 3'b001;
parameter s_TX_DATA_BITS = 3'b010;
parameter s_TX_STOP_BIT = 3'b011;
parameter s_CLEANUP    = 3'b100;

reg [2:0]  r_SM_Main    = 0;
reg [15:0] r_Clock_Count = 0;
reg [3:0]  r_Bit_Index  = 0;
reg [15:0] r_Tx_Data    = 0;
reg        r_Tx_Done    = 0;
reg        r_Tx_Active  = 0;

always @(posedge clk)
begin

    case (r_SM_Main)
        s_IDLE :
            begin
                dout <= 1'b1;    // Drive Line High for Idle
                r_Tx_Done <= 1'b0;
                r_Clock_Count <= 0;
                r_Bit_Index <= 0;

                if (start == 1'b1)
                    begin
                        r_Tx_Active <= 1'b1;
                        r_Tx_Data <= din;
                        r_SM_Main <= s_TX_START_BIT;
                    end
                else
                    r_SM_Main <= s_IDLE;
            end // case: s_IDLE

        // Send out Start Bit. Start bit = 0
        s_TX_START_BIT :
            begin
                dout <= 1'b0;

                // Wait CLKS_PER_BIT-1 clock cycles for start bit to finish
                if (r_Clock_Count < CLKS_PER_BIT-1)

```

```

begin
    r_Clock_Count <= r_Clock_Count + 1;
    r_SM_Main    <= s_TX_START_BIT;
end
else
begin
    r_Clock_Count <= 0;
    r_SM_Main    <= s_TX_DATA_BITS;
end
end // case: s_TX_START_BIT

// Wait CLKS_PER_BIT-1 clock cycles for data bits to finish
s_TX_DATA_BITS :
begin
    dout <= r_Tx_Data[r_Bit_Index];

    if (r_Clock_Count < CLKS_PER_BIT-1)
    begin
        r_Clock_Count <= r_Clock_Count + 1;
        r_SM_Main    <= s_TX_DATA_BITS;
    end
    else
    begin
        r_Clock_Count <= 0;

        // Check if we have sent out all bits
        if (r_Bit_Index < 15)
        begin
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main    <= s_TX_DATA_BITS;
        end
        else
        begin
            r_Bit_Index <= 0;
            r_SM_Main    <= s_TX_STOP_BIT;
        end
    end
end // case: s_TX_DATA_BITS

// Send out Stop bit. Stop bit = 1
s_TX_STOP_BIT :
begin
    dout <= 1'b1;

    // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
    if (r_Clock_Count < CLKS_PER_BIT-1)
    begin
        r_Clock_Count <= r_Clock_Count + 1;
        r_SM_Main    <= s_TX_STOP_BIT;
    end
    else
    begin
        r_Tx_Done    <= 1'b1;
        r_Clock_Count <= 0;
        r_SM_Main    <= s_CLEANUP;
        r_Tx_Active  <= 1'b0;
    end
end

```



```
end // case: s_Tx_STOP_BIT
```

```
// Stay here 1 clock
```

```
s_CLEANUP :
```

```
begin
```

```
    r_Tx_Done <= 1'b1;
```

```
    r_SM_Main <= s_IDLE;
```

```
end
```

```
default :
```

```
    r_SM_Main <= s_IDLE;
```

```
endcase
```

```
end
```

```
assign active = r_Tx_Active;
```

```
assign done   = r_Tx_Done;
```

```
endmodule
```

Uart Receiver كد

```
module uart_rx
(
    input    clk,
    input    din,
    output   done,
    output [15:0] dout
);

// 50000000 / 115200 = 435
parameter CLKS_PER_BIT = 435;

parameter s_IDLE      = 3'b000;
parameter s_RX_START_BIT = 3'b001;
parameter s_RX_DATA_BITS = 3'b010;
parameter s_RX_STOP_BIT = 3'b011;
parameter s_CLEANUP    = 3'b100;

reg    r_Rx_Data_R = 1'b1;
reg    r_Rx_Data   = 1'b1;

reg [15:0] r_Clock_Count = 0;
reg [3:0]  r_Bit_Index   = 0;
reg [15:0] r_Rx_Byte     = 0;
reg       r_Rx_DV        = 0;
reg [2:0]  r_SM_Main     = 0;

// Purpose: Double-register the incoming data.
// This allows it to be used in the UART RX Clock Domain.
// (It removes problems caused by metastability)
always @(posedge clk)
begin
    r_Rx_Data_R <= din;
    r_Rx_Data   <= r_Rx_Data_R;
end

// Purpose: Control RX state machine
always @(posedge clk)
begin

    case (r_SM_Main)
        s_IDLE :
            begin
                r_Rx_DV <= 1'b0;
                r_Clock_Count <= 0;
                r_Bit_Index <= 0;

                if (r_Rx_Data == 1'b0) // Start bit detected
                    r_SM_Main <= s_RX_START_BIT;
                else
                    r_SM_Main <= s_IDLE;
            end

        // Check middle of start bit to make sure it's still low
        s_RX_START_BIT :
    end
end
```

```

begin
if (r_Clock_Count == (CLKS_PER_BIT-1)/2)
begin
if (r_Rx_Data == 1'b0)
begin
r_Clock_Count <= 0; // reset counter, found the middle
r_SM_Main <= s_RX_DATA_BITS;
end
else
r_SM_Main <= s_IDLE;
end
else
begin
r_Clock_Count <= r_Clock_Count + 1;
r_SM_Main <= s_RX_START_BIT;
end
end // case: s_RX_START_BIT

```

// Wait CLKS_PER_BIT-1 clock cycles to sample serial data
s_RX_DATA_BITS :

```

begin
if (r_Clock_Count < CLKS_PER_BIT-1)
begin
r_Clock_Count <= r_Clock_Count + 1;
r_SM_Main <= s_RX_DATA_BITS;
end
else
begin
r_Clock_Count <= 0;
r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;

// Check if we have received all bits
if (r_Bit_Index < 15)
begin
r_Bit_Index <= r_Bit_Index + 1;
r_SM_Main <= s_RX_DATA_BITS;
end
else
begin
r_Bit_Index <= 0;
r_SM_Main <= s_RX_STOP_BIT;
end
end
end // case: s_RX_DATA_BITS

```

// Receive Stop bit. Stop bit = 1

s_RX_STOP_BIT :

```

begin
// Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
if (r_Clock_Count < CLKS_PER_BIT-1)
begin
r_Clock_Count <= r_Clock_Count + 1;
r_SM_Main <= s_RX_STOP_BIT;
end
else
begin
r_Rx_DV <= 1'b1;

```

```
    r_Clock_Count <= 0;  
    r_SM_Main    <= s_CLEANUP;  
end  
end // case: s_RX_STOP_BIT
```

```
// Stay here 1 clock  
s_CLEANUP :  
begin  
    r_SM_Main <= s_IDLE;  
    r_Rx_DV  <= 1'b0;  
end
```

```
default :  
    r_SM_Main <= s_IDLE;
```

```
endcase  
end
```

```
assign done  = r_Rx_DV;  
assign dout = r_Rx_Byte;
```

```
endmodule // uart_rx
```

تست Uart

```
module uart_tb ();

// Testbench uses a 50 MHz clock
// Want to interface to 115200 baud UART
// 50000000 / 115200 = 435 Clocks Per Bit.
parameter c_CLOCK_PERIOD_NS = 20;
parameter c_CLKS_PER_BIT = 435;
parameter c_BIT_PERIOD = 8700;

reg r_Clock = 0;
reg r_Tx_DV = 0;
wire w_Tx_Done;
reg [15:0] r_Tx_Byte = 0;
reg r_Rx_Serial = 1;
wire [15:0] w_Rx_Byte;

// Takes in input byte and serializes it
task UART_WRITE_BYTE;
input [15:0] i_Data;
integer ii;
begin

// Send Start Bit
r_Rx_Serial <= 1'b0;
#(c_BIT_PERIOD);
#1000;

// Send Data Byte
for (ii=0; ii<16; ii=ii+1)
begin
r_Rx_Serial <= i_Data[ii];
#(c_BIT_PERIOD);
end

// Send Stop Bit
r_Rx_Serial <= 1'b1;
#(c_BIT_PERIOD);
end
endtask // UART_WRITE_BYTE

uart_rx #(c_CLKS_PER_BIT(c_CLKS_PER_BIT)) UART_RX_INST
(.clk(r_Clock),
.din(r_Rx_Serial),
.done(),
.dout(w_Rx_Byte)
);

uart_tx #(c_CLKS_PER_BIT(c_CLKS_PER_BIT)) UART_TX_INST
(.clk(r_Clock),
.start(r_Tx_DV),
.din(r_Tx_Byte),
.active(),
```

```

.dout(),
.done(w_Tx_Done)
);

always
#(c_CLOCK_PERIOD_NS/2) r_Clock <= !r_Clock;

// Main Testing:
initial
begin

    // Tell UART to send a command (exercise Tx)
    @(posedge r_Clock);
    @(posedge r_Clock);
    r_Tx_DV <= 1'b1;
    r_Tx_Byte <= 16'hAB;
    @(posedge r_Clock);
    r_Tx_DV <= 1'b0;
    @(posedge w_Tx_Done);

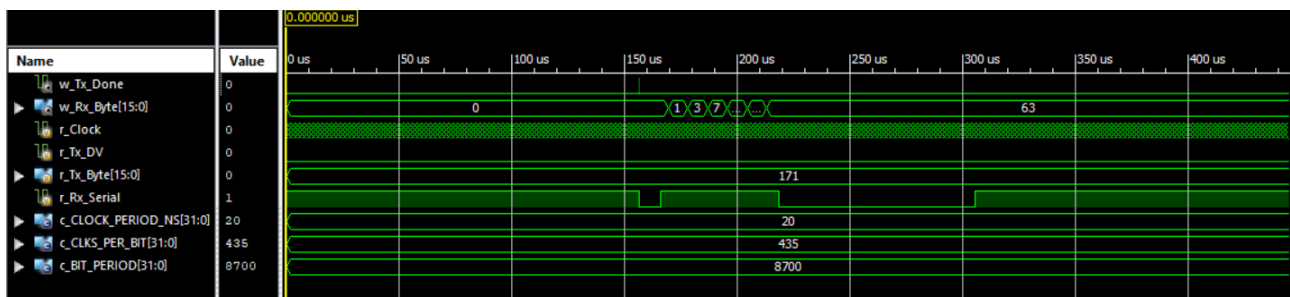
    // Send a command to the UART (exercise Rx)
    @(posedge r_Clock);
    UART_WRITE_BYTE(16'h3F);
    @(posedge r_Clock);

    // Check that the correct command was received
    if (w_Rx_Byte == 16'h3F)
        $display("Test Passed - Correct Byte Received");
    else
        $display("Test Failed - Incorrect Byte Received");

end

endmodule

```

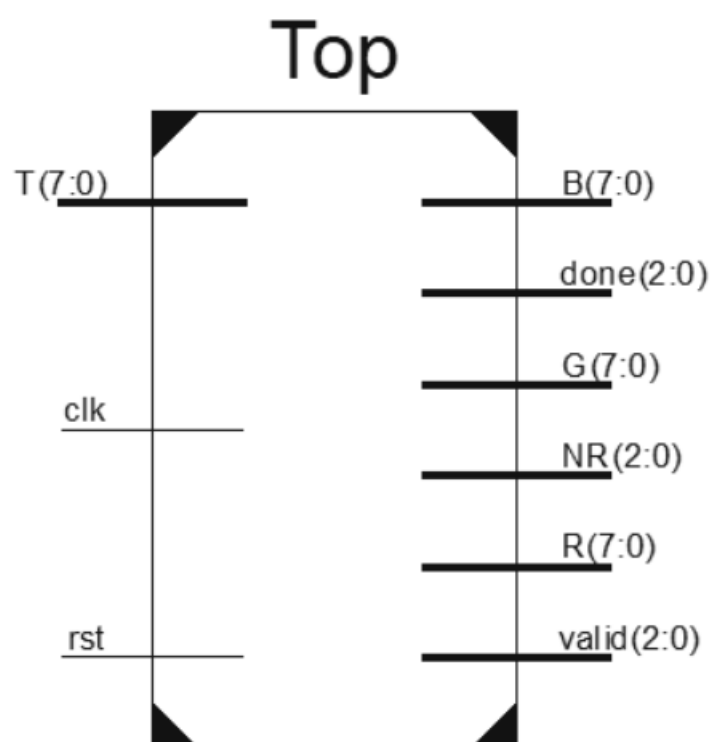


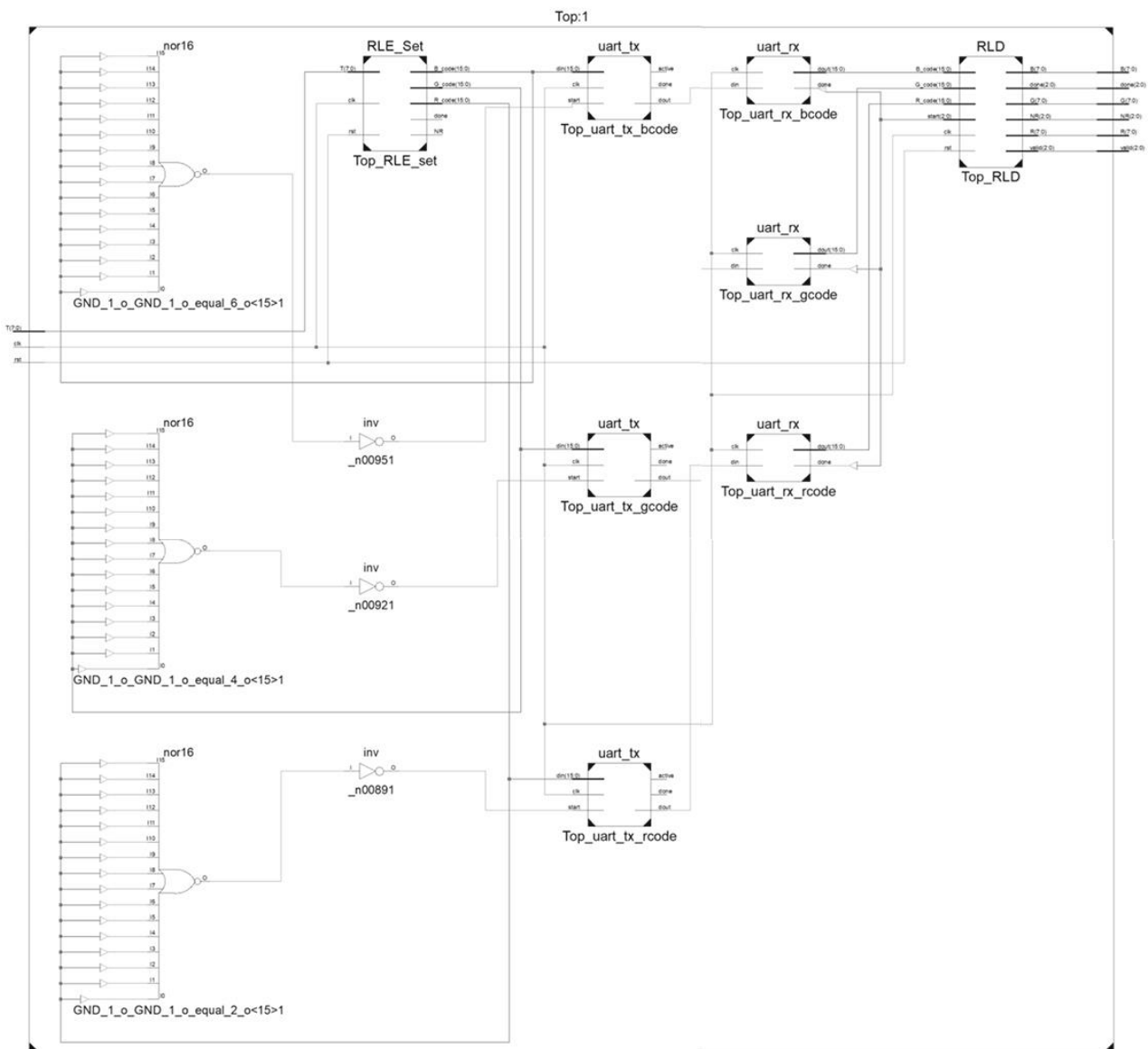
```

ISim>
# run 1ms
Test Passed - Correct Byte Received
ISim> |

```

نمای شماتیک Top






```

module Top(clk, rst, T, valid, NR, done, R, G, B);
    input clk, rst;
    input [7:0] T;
    output [2:0] valid, NR, done;
    output [7:0] R, G, B;

    wire [15:0] rle_rcode, rle_gcode, rle_bcode;

    wire tx_rcode_start, tx_gcode_start, tx_bcode_start;
    wire tx_rcode_dout, tx_gcode_dout, tx_bcode_dout;
    wire rx_rcode_done, rx_gcode_done, rx_bcode_done;
    wire [15:0] rx_rcode, rx_gcode, rx_bcode;

    wire [2:0] rld_start;

    RLE_Set Top_RLE_set (
        .clk(clk),
        .rst(rst),
        .T(T),
        .done(),
        .R_code(rle_rcode),
        .G_code(rle_gcode),
        .B_code(rle_bcode)
    );
    // 435 clks per bit
    // each code is 16 bit
    // 1 bit start 1 bit stop
    // 435 * 18 = 7830 clk per code
    defparam Top_RLE_set.HOLD_CLK = 7830;

    uart_tx Top_uart_tx_rcode (
        .clk(clk),
        .din(rle_rcode),
        .start(tx_rcode_start),
        .dout(tx_rcode_dout),
        .active(),
        .done()
    );

    uart_tx Top_uart_tx_gcode (
        .clk(clk),
        .din(rle_gcode),
        .start(tx_gcode_start),
        .dout(tx_gcode_dout),
        .active(),
        .done()
    );

    uart_tx Top_uart_tx_bcode (
        .clk(clk),
        .din(rle_bcode),
        .start(tx_bcode_start),
        .dout(tx_bcode_dout),
        .active(),
    
```

```

        .done()
    );

    uart_rx Top_uart_rx_rcode (
        .clk(clk),
        .din(tx_rcode_dout),
        .dout(rx_rcode),
        .done(rx_rcode_done)
    );

    uart_rx Top_uart_rx_gcode (
        .clk(clk),
        .din(tx_gcode_dout),
        .dout(rx_gcode),
        .done(rx_gcode_done)
    );

    uart_rx Top_uart_rx_bcode (
        .clk(clk),
        .din(tx_bcode_dout),
        .dout(rx_bcode),
        .done(rx_bcode_done)
    );

    RLD Top_RLD (
        .clk(clk),
        .rst(rst),
        .NR(NR),
        .valid(valid),
        .done(done),
        .start(rld_start),
        .R_code(rx_rcode),
        .G_code(rx_gcode),
        .B_code(rx_bcode),
        .R(R),
        .G(G),
        .B(B)
    );

    assign rld_start = {rx_bcode_done, rx_gcode_done, rx_rcode_done};

    assign tx_rcode_start = (rle_rcode == 0) ? 0 : 1;
    assign tx_gcode_start = (rle_gcode == 0) ? 0 : 1;
    assign tx_bcode_start = (rle_bcode == 0) ? 0 : 1;
endmodule

```

تست Top

```
module Top_tb;
  reg clk, rst;
  reg [7:0] T;
  wire [2:0] valid, NR, done;
  wire [7:0] R, G, B;

  integer wrf, wgf, wbf;

  Top uut (
    .clk(clk),
    .rst(rst),
    .T(T),
    .valid(valid),
    .NR(NR),
    .done(done),
    .R(R),
    .G(G),
    .B(B)
  );

  initial begin
    clk = 0;
    rst = 0;
    T = 0;
    #2 rst = 1; #7 rst = 0;
  end

  initial forever #10 clk = ~clk;

  initial begin
    #100;

    wrf = $fopen("test/decoded/R.txt", "w");

    while (~done[0]) begin
      #20;
      if (valid[0])
        $fwrite(wrf, "%0d,", R);
      if (NR[0])
        $fwrite(wrf, "\n");
    end

    $fclose(wrf);
    wrf = 0;
  end

  initial begin
    #100;

    wgf = $fopen("test/decoded/G.txt", "w");

    while (~done[1]) begin
      #20;
      if (valid[1])
        $fwrite(wgf, "%0d,", G);
```

```

    if (NR[1])
        $fwrite(wgf, "\n");
    end

    $fclose(wgf);
    wgf = 0;
end

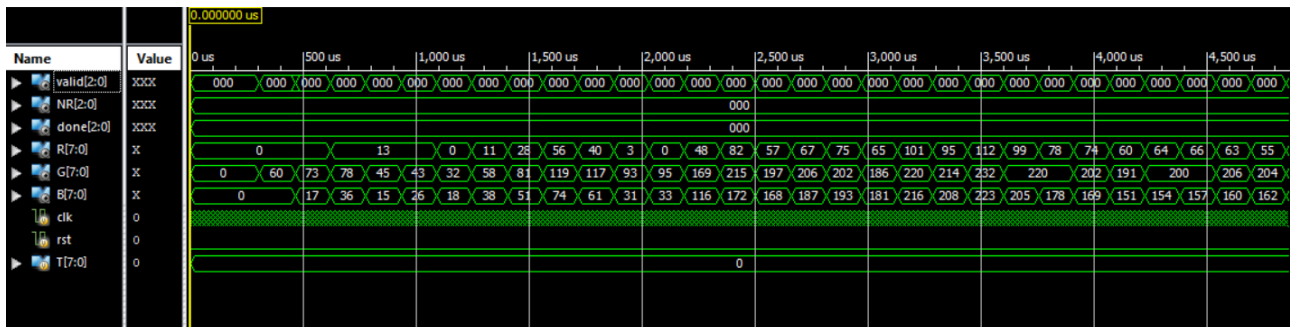
initial begin
    #100;

    wbf = $fopen("test/decoded/B.txt", "w");

    while (~done[2]) begin
        #20;
        if (valid[2])
            $fwrite(wbf, "%0d,", B);
        if (NR[2])
            $fwrite(wbf, "\n");
        end

        $fclose(wbf);
        wbf = 0;
    end
endmodule

```



کد متلب جهت تست خروجی T=0

```
close all
clear all
clc

root = '..\test\';

for i = 1:3
    rop = strcat(root, 'original\img', num2str(i), '\R.txt');
    gop = strcat(root, 'original\img', num2str(i), '\G.txt');
    bop = strcat(root, 'original\img', num2str(i), '\B.txt');

    rp = strcat(root, 'decoded\img', num2str(i), '-T0\R.txt');
    gp = strcat(root, 'decoded\img', num2str(i), '-T0\G.txt');
    bp = strcat(root, 'decoded\img', num2str(i), '-T0\B.txt');

    ro = dlmread(rop);
    ro = uint8(ro(1:128, 1:128));

    go = dlmread(gop);
    go = uint8(go(1:128, 1:128));

    bo = dlmread(bop);
    bo = uint8(bo(1:128, 1:128));

    r = dlmread(rp);
    r = uint8(r(1:128, 1:128));

    g = dlmread(gp);
    g = uint8(g(1:128, 1:128));

    b = dlmread(bp);
    b = uint8(b(1:128, 1:128));

    r_equality = isequal(r, ro);
    g_equality = isequal(g, go);
    b_equality = isequal(b, bo);

    if (r_equality && g_equality && b_equality)
        fprintf('Test %d Passed!\n', i);
    else
        fprintf('Test %d Failed!\n', i);
    end
end
```

کد متلب جهت تجزیه RGB تصویر و ساخت فایل ورودی RAM

```
close all
clear all
clc

root = '..\test\original\';

for i = 1:3
    imgp = strcat(root, 'img', num2str(i), '\img.jpg');
    rp = strcat(root, 'img', num2str(i), '\R.txt');
    gp = strcat(root, 'img', num2str(i), '\G.txt');
    bp = strcat(root, 'img', num2str(i), '\B.txt');
    coep = strcat(root, 'img', num2str(i), '\RGB.coe');

    x = imread(imgp);
    r = x(:, :, 1);
    g = x(:, :, 2);
    b = x(:, :, 3);

    dlmwrite(rp, r);
    dlmwrite(gp, g);
    dlmwrite(bp, b);

    coef = fopen(coep, 'w');
    fprintf(coef, 'memory_initialization_radix = 10;\nmemory_initialization_vector =\n');
    fclose(coef);

    dlmwrite(coep, r, '-append', 'delimiter', ' ', 'roffset', 2);
    dlmwrite(coep, g, '-append', 'delimiter', ' ', 'roffset', 2);
    dlmwrite(coep, b, '-append', 'delimiter', ' ', 'roffset', 2);

    coef = fopen(coep, 'a');
    fprintf(coef, ';');
    fclose(coef);
end
```

کد متلب جهت ترکیب RGB و ساخت تصویر

```
close all
clear all
clc

root = '..\test\decoded\img';

for i = 1:3
    pt = strcat(root, num2str(i), '-T0', '\');

    r = dlmread(strcat(pt, 'R.txt'));
    r = uint8(r(1:128, 1:128));
    g = dlmread(strcat(pt, 'G.txt'));
    g = uint8(g(1:128, 1:128));
    b = dlmread(strcat(pt, 'B.txt'));
    b = uint8(b(1:128, 1:128));

    x = cat(3, r, g, b);
    imwrite(x, strcat(pt, 'img.jpg'));

    pt = strcat(root, num2str(i), '-T30', '\');

    r = dlmread(strcat(pt, 'R.txt'));
    r = uint8(r(1:128, 1:128));
    g = dlmread(strcat(pt, 'G.txt'));
    g = uint8(g(1:128, 1:128));
    b = dlmread(strcat(pt, 'B.txt'));
    b = uint8(b(1:128, 1:128));

    x = cat(3, r, g, b);
    imwrite(x, strcat(pt, 'img.jpg'));
end
```